**Ques 1 : Create authentication service that returns JWT**

**--pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/>

<!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>jwt-auth</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>jwt-auth</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>21</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<!-- or jjwt-gson -->

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<annotationProcessorPaths>

<path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude**>**

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

**-- AuthController.java**

package com.cognizant.jwt\_auth.controller;

import com.cognizant.jwt\_auth.service.JwtService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpHeaders;

import org.springframework.http.MediaType;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthController {

    @Autowired

    private AuthenticationManager authenticationManager;

    @Autowired

    private JwtService jwtService;

    @GetMapping(value = "/authenticate", produces = MediaType.APPLICATION\_JSON\_VALUE)

    public Map<String, String> authenticate(@RequestHeader(HttpHeaders.AUTHORIZATION) String authHeader) {

        if (authHeader == null || !authHeader.startsWith("Basic ")) {

            throw new RuntimeException("Missing or invalid Authorization header");

        }

        String base64Credentials = authHeader.substring("Basic ".length());

        byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);

        String credentials = new String(decodedBytes);

        String[] values = credentials.split(":", 2);

        Authentication auth = authenticationManager.authenticate(

                new UsernamePasswordAuthenticationToken(values[0], values[1])

        );

        String token = jwtService.generateToken(values[0]);

        Map<String, String> response = new HashMap<>();

        response.put("token", token);

        return response;

    }

}

**-- JwtUtil.java**

package com.cognizant.jwt\_auth.util;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import org.springframework.stereotype.Component;

@Component

public class JwtUtil {

    private final String SECRET\_KEY = "mysecretkey123456";

    public Claims extractAllClaims(String token) {

        return Jwts.parser()

                .setSigningKey(SECRET\_KEY)

                .parseClaimsJws(token)

                .getBody();

    }

    public String extractUsername(String token) {

        return extractAllClaims(token).getSubject();

    }

    public boolean isTokenExpired(String token) {

        return extractAllClaims(token).getExpiration().before(new java.util.Date());

    }

}

**--SecurityConfig.java**

package com.cognizant.jwt\_auth.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.password.NoOpPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

public class SecurityConfig {

    @Bean

    public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {

        return http.getSharedObject(AuthenticationManagerBuilder.class)

                .inMemoryAuthentication()

                .withUser("user")

                .password("pwd")

                .roles("USER")

                .and()

                .and()

                .build();

    }

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.csrf().disable()

                .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())

                .httpBasic()

                .and()

                .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        return http.build();

    }

    @Bean

    public PasswordEncoder passwordEncoder() {

        // Use NoOpPasswordEncoder only for testing/demo!

        return NoOpPasswordEncoder.getInstance();

    }

}

**--JwtAuthApplicationTests.java**

package com.cognizant.jwt\_auth;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class JwtAuthApplicationTests {

    @Test

    void contextLoads() {

    }

**}**

**--JwtService.java**

package com.cognizant.jwt\_auth.service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Service;

import javax.crypto.SecretKey;

import java.util.Date;

@Service

public class JwtService {

    private final SecretKey secretKey = Keys.secretKeyFor(SignatureAlgorithm.HS256);

    public String generateToken(String username) {

        long now = System.currentTimeMillis();

        long expiry = 1000 \* 60 \* 60; // 1 hour expiration

        return Jwts.builder()

                .setSubject(username)

                .setIssuedAt(new Date(now))

                .setExpiration(new Date(now + expiry))

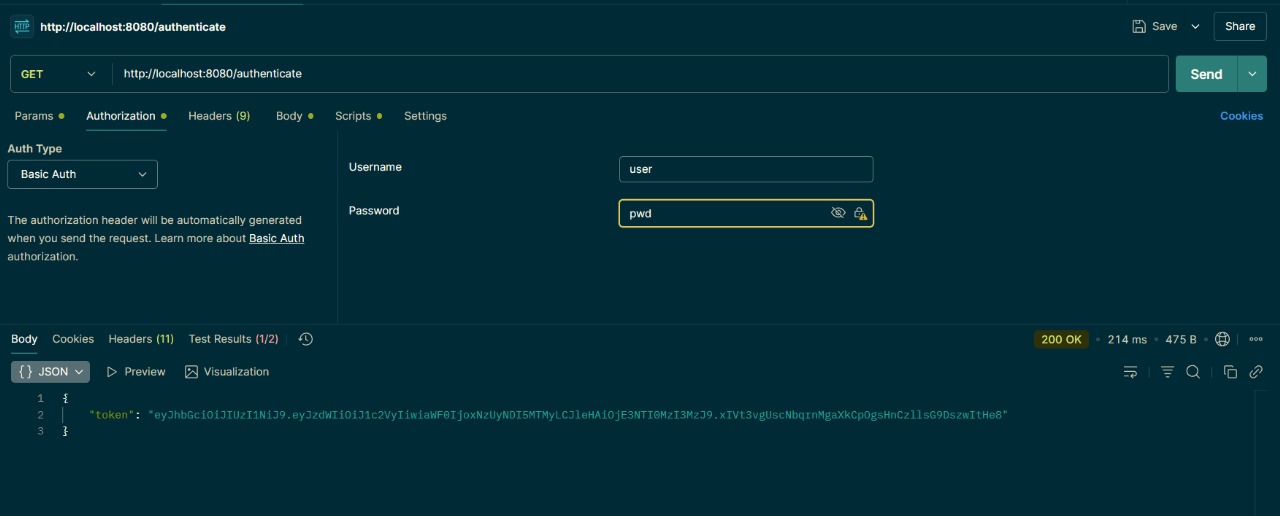
                .signWith(secretKey, SignatureAlgorithm.HS256)

                .compact();

    }

}

**Output :**

****